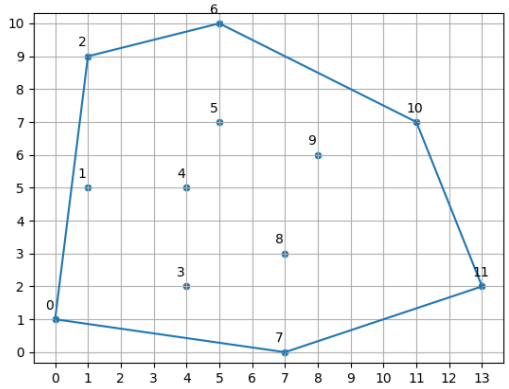
Enveloppe convexe d'un nuage de points

Ce problème a pour objectif le calcul de l'enveloppe convexe d'un nuage de points du plan. L'enveloppe convexe d'un ensemble fini $P \subset \mathbb{R}^2$ est le plus petit polygone convexe contenant P, par exemple :



Les sommets de ce polygone appartiennent à P. Dans toute la suite on supposera que le nuage de points P contient $n \geq 3$ points numérotés de 0 à $n-1$ placés en position générale, c'est-à-dire qu'il n'y a pas 3 points distincts alignés. Cette hypothèse va permettre de simplifier les calculs en ignorant les cas gênants, notamment la présence de 3 points alignés sur le bord de l'enveloppe convexe. Les coordonnées du point `i` dans un repère orthonormé direct sont notés x_i et y_i.
Un nuage P de n points est donné par le tableau trié `[[`x_0`, `y_0`],`$\cdots$`,[`x_n`, `y_n`]]`.
Ainsi, sur l'exemple représenté ci-dessus, P vaut :
`[[0,1],[1,5],[1,9],[4,2],[4,5],[5,7],[5,10],[7,0],[7,3],[8,6],[11,7],[13,2]]`.

Rappels sur les extensions à utiliser

Graphiques

```
import matplotlib.pyplot as plt

plt.plot(lx, ly) # préaffiche la ligne polygonale composée des points
# dont les abscisses sont dans lx et les ordonnées dans ly

plt.scatter(lx, ly, s = 20) # préaffiche le nuage de points composé des points
# dont les abscisses sont dans lx et les ordonnées dans ly
#s donne la taille du point

plt.show() # affiche une figure
```

Tirage au hasard

```
import random as rd
rd.random()# renvoie un flottant tiré au hasard dans [0,1[ (uniformément)
```

I Tri du nuage de points

On suppose que P est initialement donné sous la forme suivante :
`P = [[1,5],[11,7],[8,6],[7,0],[5,10],[7,3],[1,9],[0,1],[4,5],[13,2],[4,2],[5,7]]`.
On souhaiterait le trier selon les abscisses et, pour une même abscisse, selon les ordonnées.

Q1 Écrire une fonction `tri_rapide_x(P)` qui trie P selon les abscisses, sans tenir compte des ordonnées. *Par exemple,* `tri_rapide_x(P)` renvoie
`[[0,1],[1,5],[1,9],[4,5],[4,2],[5,10],[5,7],[7,0],[7,3],[8,6],[11,7],[13,2]]`.

Le nombre de points ayant la même abscisse étant supposé petit, on préfère un tri par insertion selon les ordonnées.

Q2 Écrire une fonction `tri_insertionx_y(P)` qui trie P selon les ordonnées, sans tenir compte des abscisses. *Par exemple,* `tri_insertionx_y([[1,4],[1,0],[1,3],[2,2]]))` renvoie `[[1, 0], [2, 2], [1, 3], [1, 4]]`.

Q3 Écrire une fonction `tri_insertion_y(P)` qui, à partir d'une liste triée selon les abscisses, trie chacune des sous-listes de même abscisse selon les ordonnées.
Par exemple, `tri_insertion_y(tri_rapide_x(P))` renvoie
`[[0,1],[1,5],[1,9],[4,2],[4,5],[5,7],[5,10],[7,0],[7,3],[8,6],[11, 7],[13,2]]`.

II Calcul préliminaires

Q4 Écrire une fonction `plusBas(P)` qui renvoie l'indice j du point le plus bas (c'est-à-dire de plus petite ordonnée) parmi les points du nuage P. En cas d'égalité, la fonction devra renvoyer l'indice du point de plus petite abscisse parmi les points les plus bas.
Sur le même schéma que celui de la fonction précédente, écrire trois fonctions `plusHaut(P)`, `plusGauche(P)`, `plusDroite(P)` qui renvoient l'indice j respectivement du point le plus haut, du point le plus à gauche et du point le plus à droite de P.
Par exemple, sur le nuage donné en introduction :
`plusBas(P)` $\longrightarrow$ `7`, `plusHaut(P)` $\longrightarrow$ `6`, `plusGauche(P)` $\longrightarrow$ `0`, `plusDroite(P)` $\longrightarrow$ `11`.

Q5 Écrire une fonction `densite_rec(P)` qui renvoie la densité rectangulaire du nuage de points P, c'est-à-dire le quotient

$$\frac{n}{(x_{max}-x_{min})(y_{max}-y_{min})}$$

où x_{min} et x_{max} sont respectivement la plus petite et la plus grande abscisse des points de P, où y_{min} et y_{max} sont respectivement la plus petite et la plus grande ordonnée des points de P, et où n est le nombre de points de P.
Par exemple, sur le nuage donné en introduction :
`densite_rec(P)` $\longrightarrow$ `0.09230769230769231`.

Q6 Un triangle ABC sera dit direct si la mesure principale de l'angle $(\overrightarrow{AB},\overrightarrow{AC})$ appartient à $]0,\pi[$ et indirect si la mesure principale de l'angle $(\overrightarrow{AB},\overrightarrow{AC})$ appartient à $]-\pi,0[$.
On admet que ABC est direct si et seulement si $det(\overrightarrow{AB},\overrightarrow{AC})>0$ et que ABC est indirect si et seulement si $det(\overrightarrow{AB},\overrightarrow{AC})<0$ (les déterminants sont calculés dans la base canonique du plan $\mathbb{R}^2$).
Écrire une fonction `orientation(P, i, j, k)` qui renvoie renvoie `1` si le triangle formé des points numérotés `i,j,k` de P est orienté positivement, qui renvoie `-1` si le triangle est indirect, et qui renvoie `0` si le triangle est aplati (cette dernière situation ne devrait pas se rencontrer si le nuage de points ne contient pas trois points alignés).
Par exemple, sur le nuage donné en introduction :
`orientation(P, 1, 3, 5)` $\longrightarrow$ `1`.

III Algorithme de Jarvis, dit « du paquet cadeau »

L'algorithme de Jarvis, qui date de 1973, consiste à envelopper peu à peu le nuage de points P dans une sorte de « paquet cadeau », qui à la fin du processus est exactement le bord de l'enveloppe convexe de P.

- Le point de plus petite ordonnée du nuage P est le premier sommet du polygone enveloppe convexe.
- Supposons la construction du polygone enveloppe convexe amorcée jusqu'au sommet `i`. Le sommet qui suit `i` dans l'enveloppe convexe est l'unique point `j`$\neq$`i` du nuage P tel, que pour tout point `k` du nuage P distinct de `i` et `j`, le triangle `i`,`j`,`k` est direct. On admet que l'hypothèse faite en début d'énoncé (absence de triplet de points alignés) entraîne l'unicité du point à insérer.
- L'algorithme prend fin lorsque le sommet suivant `i` est le premier sommet du polygone.

Q7 Écrire une fonction `sommetSuivant(P, i)` qui renvoie le numéro `j` du point de P qui est le sommet suivant de `i` dans le polygone enveloppe convexe de P. Quelle est la complexité de cette fonction ?

Q8 On remarque que si, pour un point `j`, il existe un point `k` tel que le triangle `i`,`j`,`k` soit indirect, alors `j` est plus grand que `k`. Utiliser cette remarque pour écrire une fonction `sommetSuivant2(P, i)` de complexité linéaire $\mathcal{O}(n)$ qui a le même résultat que la fonction précédente (sauter cette question si l'on a déjà obtenu une complexité linéaire à la question précédente).

Q9 Écrire une fonction `enveloppe(P)` qui renvoie l'enveloppe convexe de P donné sous la forme de la liste des numéros des points de P qui sont les sommets (dans l'ordre) du polygône enveloppe convexe.
Par exemple, si P correspond à l'exemple dessiné en introduction du sujet :
`enveloppe(P)` $\longrightarrow$ `[7, 11, 10, 6, 2, 0]`.

IV Analyse du résultat

Q10 Écrire une fonction `grapheEnveloppe(P)` qui renvoie le graphique du nuage de points P et de son polygone enveloppe convexe (voir le rappel sur les extensions à utiliser).

Q11 Écrire une fonction `nuageHasard(n)` qui renvoie un nuage de n points tirés au hasard dans le carré $[0,1] \times [0,1]$.

Q12 On voudrait estimer le nombre de sommets que comporte l'enveloppe convexe d'un nuage P de n points. Pour cela, on tire au hasard m nuages de n points avec la fonction précédente (on suppose que la probabilité d'obtenir 3 points alignés dans l'un des nuages est négligeable). Écrire une fonction `histo(n, m)` qui renvoie l'histogramme `NB` des nombres de points de l'enveloppe convexe des nuages tirés au hasard, c'est-à-dire la liste
`[NB[0], NB[1], ⋯, NB[`n`]]` où `NB[`i`]` est le nombre de nuages dont l'enveloppe comporte i points.

Q13 En déduire une fonction `moyenne(n, m)` qui renvoie la moyenne du nombre de points de l'enveloppe convexe de m nuages de n points tirés au hasard.

Q14 On conjecture que le nombre moyen de points de l'enveloppe convexe d'un nuage de n points du carré $[0,1] \times [0,1]$ tirés au hasard est proportionnel à $ln(n)$. Proposer un moyen de le mettre en évidence et d'estimer la constante de proportionnalité.

Corrigé

Q1

```
16 def tri_rapide_x(P):
17     if len(P) <= 1:
18         return P
19     else:
20         pivot = P[0]
21         G, D = [], []
22         for i in range(1, len(P)):
23             e = P[i]
24             if e[0] < pivot[0]:
25                 G.append(e)
26             else:
27                 D.append(e)
28         return tri_rapide_x(G) +  [P[0]] + tri_rapide_x(D)
```

Q2

```
31 def insertion_y(M, x):
32     for i in range(len(M)):
33         if x[1] <= M[i][1]:
34             return M[:i] + [x] + M[i:]
35     return M + [x]
36 
37 def tri_insertionx_y(L):
38     M = []
39     for x in L :
40         M = insertion_y(M, x)
41     return M
```

Q3

```
44 def tri_insertion_y(P):
45     Q, L = [], []
46     for i in range(len(P)):
47         L.append(P[i])
48         if (i+1 < len(P) and P[i][0] != P[i+1][0]) or i+1 == len(P):
49             Q = Q + tri_insertionx_y(L)
50             L = []
51     return Q
52 
53 P = [[1, 5], [11, 7], [8, 6], [7, 0], [5, 10], [7, 3], [1, 9], [0, 1],
       [4, 5], [13, 2], [4, 2], [5, 7]]
54 P = tri_insertion_y(tri_rapide_x(P))
```

Q4

```
def plusBas(P):
    m_y = float('inf') # plus petite ordonnée
    i_y = 0
    for i in range(len(P)):
        x, y = P[i]
        if y < m_y:
            m_y = y
            i_y = i
    return i_y

assert plusBas(P) == 7

#
def plusHaut(P):
    M_y = -float('inf') # plus grande ordonnée
    for i in range(len(P)):
        x, y = P[i]
        if y > M_y:
            M_y = y
            i_y = i
    return i_y

assert plusHaut(P) == 6

#
def plusGauche(P):
    m_x = float('inf') # plus petite abscisse
    for i in range(len(P)):
        x, y = P[i]
        if x < m_x:
            m_x = x
            i_x = i
    return i_x

assert plusGauche(P) == 0

#
def plusDroite(P):
    M_x = -float('inf') # plus grande abscisse
    for i in range(len(P)):
        x, y = P[i]
        if x > M_x:
            M_x = x
            i_x = i
    return i_x

assert plusDroite(P) == 11
```

Q5

```
def densite_rec(P):
    n = len(P)
    xMax = P[plusDroite(P)][0]
    xMin = P[plusGauche(P)][0]
    yMax = P[plusHaut(P)][1]
    yMin = P[plusBas(P)][1]
    return n/((xMax-xMin)*(yMax-yMin))

assert densite_rec(P) == 0.09230769230769231
```

Q6

```
def determinant(vec1, vec2):
    xVec1, yVec1 = vec1
    xVec2, yVec2 = vec2
    return xVec1 * yVec2 - xVec2 * yVec1

def vecteur(point1, point2):
    xPoint1, yPoint1 = point1
    xPoint2, yPoint2 = point2
    return [xPoint2 - xPoint1, yPoint2 - yPoint1]

def orientation(P, i, j, k):
    if determinant(vecteur(P[i], P[j]), vecteur(P[i], P[k])) == 0:
        return 0
    else:
        det = determinant(vecteur(P[i], P[j]), vecteur(P[i], P[k]))
        return int(det / abs(det))
```

Q7

```
def sommetSuivant(P, i):
    n = len(P)
    for j in range(n):
        if j != i: # ce test évite une boucle inutile sur k
            k = 0
            while k < n and (k == i or k == j or orientation(P, i, j, k)
    > 0):
                k = k + 1
            if k == n:
                return j

assert sommetSuivant(P, 7) == 11
assert sommetSuivant(P, 11) == 10
assert sommetSuivant(P, 10) == 6
assert sommetSuivant(P, 6) == 2
assert sommetSuivant(P, 2) == 0
assert sommetSuivant(P, 0) == 7

# ou
def sommetSuivant(P, i):
    n = len(P)
    for j in range(n):
        if j != i:
            direct = True
            for k in range(n):
                if k != i and k != j and orientation(P, i, j, k) <= 0:
                    direct = False
            if direct:
                return j
```

Q8

```
def sommetSuivant2(P, i):
    n = len(P)
    k = 0
    j = k
    for k in range(n) :
    #on fait croître k et dès que une orientation n'est plus bonne, j
   prend la valeur de k
        if k != i and orientation(P, i, j, k) != 1:
            j = k
    return j
```

```
174
175 assert sommetSuivant2(P, 7) == 11
176 assert sommetSuivant2(P, 11) == 10
177 assert sommetSuivant2(P, 10) == 6
178 assert sommetSuivant2(P, 6) == 2
179 assert sommetSuivant2(P, 2) == 0
180 assert sommetSuivant2(P, 0) == 7
```

Q9

```
183 def enveloppe(P):
184     i0 = plusBas(P)
185     env = [i0]
186     sommet = sommetSuivant(P, i0)
187     while sommet != i0:
188         env.append(sommet)
189         sommet = sommetSuivant(P, sommet)
190     return env
```

Q10

```
193 import matplotlib.pyplot as plt
194 import numpy
195
196 def grapheEnveloppe(P):
197     fig = plt.figure()
198     # grille avec des lignes à chaque unité
199     ax = fig.gca()
200     ax.set_xticks(numpy.arange(0, 14, 1))
201     ax.set_yticks(numpy.arange(0, 14, 1))
202     plt.grid()
203     # abscisses et ordonnées des points
204     lx = [P[i][0] for i in range(len(P))]
205     ly = [P[i][1] for i in range(len(P))]
206     # nom sur les points
207     for i in range(len(lx)):
208          plt.annotate(i, (lx[i], ly[i]), xytext=(lx[i]-0.3, ly[i]+0.3))
209     plt.scatter(lx, ly, s = 20)
210     # enveloppe convexe
211     env = enveloppe(P)
212     lxe = [P[env[i]][0] for i in range(len(env))]
213     lye = [P[env[i]][1] for i in range(len(env))]
214     lxe.append(P[env[0]][0])
215     lye.append(P[env[0]][1])
216     plt.plot(lxe, lye)
217     plt.axis("equal")
218     plt.show()
219
220 grapheEnveloppe(P)
```

Q11

```
223 import random as rd
224
225 def nuageHasard(n):
226     return [[rd.random(), rd.random()] for _ in range(n)]
```

Q12

```
229 def histo(n, m):
230     return [len(enveloppe(nuageHasard(n))) for _ in range(m)]
```

Q13

```
233 def moyenne(n, m):
234     H = histo(n, m)
235     s = 0
236     for hist in H:
237         s = s + hist
238     return s/m
```

Q14

```
241 from math import log
242 
243 m = 100
244 N = 30
245 
246 plt.figure()
247 lx = [log(n) for n in range(2,N)]
248 ly = [moyenne(n,m) for n in range(2,N)]
249 plt.plot(lx,ly)
250 plt.show()
251 
252 #
253 
254 M = []
255 for n in range(N):
256     M.append(moyenne(n,m)/log(n))
257 print(sum(M)/N)
```